Tutorial 9

Homologia Dinâmica

BIZ0433 - INFERÊNCIA FILOGENÉTICA: FILOSOFIA, MÉTODO E APLICAÇÕES.

Conteúdo

Objetivo

O objetivo deste tutorial é introduzir o conceito de otimização direta e homologia dinâmica. O tutorial apresenta uma introdução ao uso de POY, programa no qual tais conceitos estão implementados operacionalmente. A introdução ao uso de POY explora os conceitos básicos relacionados à sintaxe de comandos do programa, o uso de linguagem de *scripts* a e implementação de funções de custo durante o processo de otimização direta. Os arquivos associados a este tutorial estão disponíveis no GitHub. Você baixar todos os tutoriais com o seguinte comando:

```
svn checkout https://github.com/fplmarques/cladistica/trunk/tutorials/
```

9.1 Contextualização

No tutorial anterior (Tutorial 8) observamos a interdependência de parâmetros de alinhamento e árvores-guias no alinhamento e consequentemente em inferência filogenética. Saber o que deve ser considerado um "bom" alinhamento é impossível. Há uma série de artigos comparando métodos de alinhamento [veja 1:145, e referências citadas]. Estas comparações levam em consideração as distâncias entre alinhamentos "reais" (*i.e.*, dados simulados) daqueles inferidos por programas de alinhamento múltiplos [*i.e.*, 2] ou performance de acordo com uma função objetiva qualquer (*e.g.*, SP[1] ou custo da topologia) [*e.g.*, 3].

Wheeler [1] argumenta que alinhamentos "reais" provavelmente nunca serão encontrados na natureza e que, portanto, comparações com dados simulados são inapropriadas. Contrariamente, Ogden & Rosemberg [2:190] consideram inapropriado comparar custos de topologias geradas por diferentes alinhamentos. Isso porque alinhamentos distintos postulam diferentes proposições de homologia e assim devem ser considerados matrizes de dados distintas.

Em inferência filogenética, o foco central de qualquer procedimento analítico é identificar a(s) melhor(es) topologia(as) de acordo com o critério de otimalidade selecionado pelo investigador. Métodos que geram soluções melhores para este problema devem ser favorecidos [3]. Como a escolha de topologias é feita pelo ordenamento destas hipóteses de acordo com alguma função objetiva (*e.g.*, distância patrística em parcimônia), o melhor que um método pode fazer é otimizar custo de topologias – o chamado *Tree Alignment Problem* (**TAP**) – [4]. Como definido por Wheeler [1:133], **TAP** foi originalmente descrito como um problema no qual, dado uma topologia, objetiva-se identificar o alinhamento que minimiza o seu custo. Neste processo, o resultado final é o alinhamento implícito [senso 5], resultado dos estados de caráter atribuídos aos nós (vértices) da topologia. Devemos considerar que há uma complexidade combinatória relacionada às possíveis atribuições de estados de caráter no nós da topologia. De fato, pode existir um número exponencial de soluções dado uma topologia. Veja o exemplo na Figura 9.1.

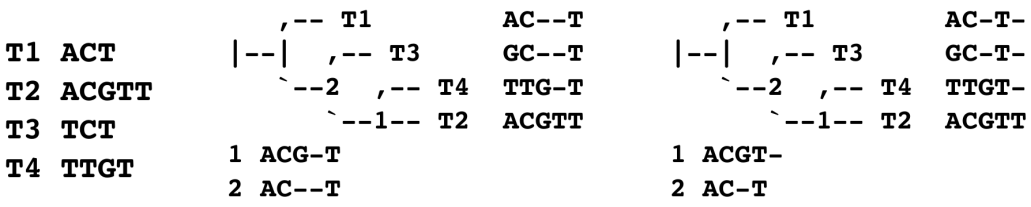

Figura 9.1: Exemplo de *Tree Alignment*. Considere as sequências à esquerda. Há uma única topologia para estas sequências com o custo de 5 transformações. No entanto há dois alinhamentos implícitos relacionados a esta topologia. Para cada um deles, a atribuição dos estados nos nós é diferente.

Análises filogenéticas de dados moleculares apresentam alguns desafios além daqueles inerentes da complexidade associada ao exame ou exploração do espaço de topologias que é uma função determinada pelo número de terminais – como vimos no Tutorial 3. Dados moleculares podem variar não somente em números de elementos, isto é, estados de caracteres, mas também em

[1] *sum-of-pairs* de um alinhamento qualquer é a soma dos custos dos alinhamentos par-a-par existentes neste alinhamento (maiores detalhes em http://cs124.cs.ucdavis.edu/lectures/multextra.pdf

tamanho (veja Tutorial 8). Nestes casos, tradicionalmente é feito um alinhamento a partir do qual se procede com as análises de inferência filogenética. Este alinhamento estabelece as correlações entre estados de caráter em cada coluna, considerados como proposições de homologia entre os pares de base de cada sequência dentro do contexto de homologia estática. Como visto anteriormente (seção 8.4.1 do Tutorial 8), esquemas putativos de homologia (alinhamentos) variam entre diferentes topologias utilizadas durante o alinhamento. Também foi observado que estes cenários putativos de alinhamento geram topologias com custos distintos. O conceito de homologia dinâmica [senso 6] integra alinhamento e busca de topologias em um único procedimento analítico conhecido como otimização direta [*direct optimization*; 7]. Ao integrar esses processos analíticos, conjugamos dois níveis de complexidades, um associado ao número de topologia e o outro associado ao número de alinhamentos possíveis.

Neste tutorial iremos explorar alguns conceitos práticos e teóricos associados à otimização direta de sequências nucleotídicas implementadas em POY [8, 9]. De nenhuma maneira este tutorial irá explorar toda a funcionalidade de POY e o interessado deve explorar os tutoriais disponíveis na documentação do programa (veja materiais associados a esse tutorial e literatura referenciada no final deste tutorial).

9.2 Introdução ao POY

9.2.1 Etapas de execução em POY

O POY é um programa extremamente versátil. Parte desta versatilidade está na possibilidade de analisar simultaneamente diferentes fontes de dados (*i.e.*, sequências nucleotídicas, matrizes morfológicas, entre outras), mas também na forma de interpretação e execução de *scripts*. Este último componente pode ser a parte mais complexa do uso de POY e o aluno interessado em possuir melhor domínio da linguagem de *script* de POY deve consultar a Seção 3.1 da documentação do programa (`tutorial_9/literature/poy_commands.pdf`). A execução mais simples de POY requer alguns componentes primários. O primeiro deles é a leitura dos arquivos de dados. O segundo, é a implementação da função de custo, ou seja, os parâmetros de alinhamento. Essa etapa é opcional caso o usuário queira utilizar os valores de *default* do programa – custo 1 para todas as transformações. Uma vez cumprida estas etapas, o usuário inicia a busca, seleciona a topologia mais curta e reporta o resultado. Vamos ver como isso é feito na prática.

Em um terminal, execute:

```
$ poy5.1.2a
```

Você deverá obter:

```
alan@turing: $ poy5.1.2a
Information : Welcome to POY 5.1.1 (2141b0ef4b2f+)
Compiled on Thu Jul 23 10:03:53 EDT 2020 with parallel off, interface flat,
likelihood on, and concorde off.
Copyright (C) 2011, 2012, 2013, 2014 Andres Varon, Nicholas Lucaroni, Lin
Hong, Ward Wheeler, and the American Museum of Natural History.
POY 5.1.1 comes with ABSOLUTELY NO WARRANTY; This is free software, and you
```

```
are welcome to redistribute it under the GNU General Public License Version
2, June 1991.


Information : Setting random seed value to 1431363032
Information :
Information :
Information :
Information : For help, type help().

        Enjoy!
Information :
          Trees:
            Storing 0 trees
          Cost Mode: Normal Direct Optimization
poy>
```

Nosso primeiro passo é fazer a leitura do arquivo se dados. Nesse exemplo iremos usar o arquivo `seqdata1.fas`. No *prompt* de POY execute:

```
poy> read("seqdata1.fas")
```

você deverá obter:

```
poy> read("seqdata1.fas")
Information : Reading file seqdata1.fas of type input sequences
Information :
The file seqdata1.fas contains sequences of 10 taxa, each sequence holding 1
fragment.
Status : Loading Trees Finished
Information :             Trees:
            Storing 0 trees
          Cost Mode: Normal Direct Optimization

poy>
```

Neste exemplo, POY informa que leu o arquivo `seqdata1.fas` e que o mesmo possui um único fragmento.

Nosso próximo passo é implementar a busca; pois iremos usar as funções de custo internas do programa. Iremos fazer isso utilizando os métodos convencionais para explorar o espaço de topologias como fizemos inicialmente em TNT (*i.e.*, **RAS+SWAP**), veja seção 4.7 do Tutorial 4). Para implementar este tipo de busca são necessários dois comandos. O primeiro é o comando `build()`, que constrói árvores de Wagner à medida em que calcula o custo utilizando o algoritmo de Needleman-Wunsch [10]. O segundo é o comando `swap()` que executa o refinamento por SPR e TBR e, para cada rearranjo, calcula o custo da topologia da mesma forma que o fez na construção da árvore de Wagner. Vejamos isso na prática. No prompt de POY, execute:

```
poy> build(100)
```

Observe que o comando `build()` requer um argumento que, neste caso, foi um valor numérico que é interpretado pelo POY como o número de RAS (*random addition sequence* que deverá ser executado. Após a execução deste comando, você deverá obter:

```
poy> build(100)
Status : Wagner : 2 of 10 – Wagner tree with cost 18.
Status : Wagner : 3 of 10 – Wagner tree with cost 34.
Status : Wagner : 4 of 10 – Wagner tree with cost 59.
Status : Wagner : 5 of 10 – Wagner tree with cost 75.
Status : Wagner : 6 of 10 – Wagner tree with cost 104.
Status : Wagner : 7 of 10 – Wagner tree with cost 114.
Status : Wagner : 8 of 10 – Wagner tree with cost 138.
Status : Wagner : 9 of 10 – Wagner tree with cost 152.
Status : Wagner Finished
Status : Building Wagner Tree Finished
Status : Running Pipeline : 1 of 100 – Estimated finish in 3 s
Status : Wagner : 2 of 10 – Wagner tree with cost 24.
Status : Wagner : 3 of 10 – Wagner tree with cost 39.
Status : Wagner : 4 of 10 – Wagner tree with cost 64.
Status : Wagner : 5 of 10 – Wagner tree with cost 88.
Status : Wagner : 6 of 10 – Wagner tree with cost 101.
Status : Wagner : 7 of 10 – Wagner tree with cost 112.
Status : Wagner : 8 of 10 – Wagner tree with cost 129.
Status : Wagner : 9 of 10 – Wagner tree with cost 155.
Status : Wagner Finished
Status : Building Wagner Tree Finished
Status : Running Pipeline : 2 of 100 – Estimated finish in 2 s
...
Status : Wagner Finished
Status : Wagner build : 100 of 100 – Estimated finish in 0 s
Status : Wagner : 2 of 10 – Wagner tree with cost 33.
Status : Wagner : 3 of 10 – Wagner tree with cost 56.
Status : Wagner : 4 of 10 – Wagner tree with cost 72.
Status : Wagner : 5 of 10 – Wagner tree with cost 93.
Status : Wagner : 6 of 10 – Wagner tree with cost 113.
Status : Wagner : 7 of 10 – Wagner tree with cost 129.
Status : Wagner : 8 of 10 – Wagner tree with cost 138.
Status : Wagner : 9 of 10 – Wagner tree with cost 148.
Status : Wagner Finished
Status : Wagner build : 101 of 100 – Estimated finish in 0 s
Status : Wagner build Finished
Information :
        Trees:
          Storing 100 trees with costs 153. to 165.
          Best cost in 9 trees
        Cost Mode: Normal Direct Optimization
poy>
```

Os resultados acima indicam que o POY contruiu 100 árvores de Wagner cujos custos variam de 153 a 165[2]. Dentro destas 9 delas possuem o menor custo.

Nossa próxima etapa será o refinamento destas topologias implementando rearranjos via SPR e TBR. O comando do POY para essas estratégias de refinamento é o `swap()`. Esse comando implementará os algoritmos de refinamento para todas as topologias contidas na memória. A forma mais simples de execução desse comando é a seguinte:

```
poy> swap()
```

cujo resultado seria:

```
poy> swap()
Status : Tree search : 1 of 100 --
Status : TBR : 174 Searching
Status : TBR Finished
...
Status : SPR : 157 157.
Status : SPR : 155 155.
Status : SPR : 154 154.
Status : SPR Finished
Status : Alternate : 0 Performing TBR swapping
Status : Single TBR Finished
Status : Alternate Finished
Status : Tree search : 100 of 100 – Estimated finish in 0 s
Status : Alternate : 0 Beginning search
Status : Alternate : 0 SPR search
Status : SPR : 153 153.
Status : SPR Finished
Status : Alternate : 0 Performing TBR swapping
Status : Single TBR Finished
Status : Alternate Finished
Status : Tree search Finished
Information :
        Trees:
          Storing 100 trees with costs 153. to 160.
          Best cost in 38 trees
        Cost Mode: Normal Direct Optimization
poy>
```

Após o refinamento, POY manteve 100 topologias que variaram de 153 a 160 em custo dentre as quais 38 possuem o menor custo.

Como dentro do conjunto de topologias mantidas na memória de POY há árvores subótimas, nosso próximo passo é selecionar aquelas que nos interessam. Para a seleção de topologias **únicas** e **ótimas**, utilizamos os parâmetros de *default* do comando `select()`. No terminal do POY execute:

[2] Estes números podem variar de execução a execução dado a aleatoriedade associada ao algoritmo.

```
poy> select()
```

você deverá obter:

```
poy> select()
Information :
        Trees:
         Storing 3 trees with costs 153. to 153.
         Best cost in 3 trees
        Cost Mode: Normal Direct Optimizationn
poy>
```

O resultado de POY indica que foram encontradas 3 topologias com o custo de 153. Observe que antes de selecionarmos as topologias **únicas** e **ótimas** do conjunto de topologias que compilamos após o refinamento – na etapa anterior –, haviam 38 árvores com o custo de 153. O resultado após implementarmos o comando `select()` significa que dentro deste conjunto de 38 topologias, apenas três delas eram diferentes uma das outras.

Finalmente, vamos verificar nossos resultados, ou seja, as topologias obtidas. POY imprime os resultados ou informações sobre os arquivos de entrada pelo comando `report()`. Há várias formas de obter as topologias encontradas por POY, vejamos algumas delas. No prompt de POY, execute:

```
poy> report(asciitrees)
```

Você deverá obter as topologias encontradas impressa no terminal no formato visual com caracteres ASCII. Se você executar:

```
poy> report(trees:(total))
```

Você deverá obter as topologias encontradas impressa no terminal em formato parentético (*i.e., newick format*) e seus respectivos custos entre colchetes.

Finalmente, o comando `report()` permite que você direcione os resultados para arquivos e que você faça isso utilizando um ou mais arguentos. Por exemplo, se você executar:

```
poy> report("newicktrees.tre",trees,"asciitrees.txt",asciitrees)
```

Você deverá obter dois arquivos em seu diretório de trabalho. O primeiro é o arquivo `newicktrees.tre` que conterá as topologias selecionadas em formato parentético. O segundo é o arquivo texto `asciitrees.txt` que conterá as topologias selecionadas em formato ASCII.

Finalmente, para sair de POY você deve digitar `exit()`.

9.2.2 *Scripts* DE EXECUÇÃO EM POY

A maneira mais efetiva de utilizar o POY é utilizando *scripts*. Os *scripts* de POY são arquivos textos contendo os comandos de execução seguindo a estrutura analítica que acabamos de fazer. Vamos repetir o que acabamos de fazer utilizando essa forma de execução.

Exercicio 9.1

Neste exercício você deverá executar o POY utilizando um *script*.

i. Crie um arquivo texto chamado `meu_script1.poy` com o seguinte conteúdo:

Listing 9.1: conteúdo do arquivo meu_script1.poy

```
read("seqdata1.fas")
set(root:"Taxon1")
build(100)
swap()
select()
report("seqdata1_do.tre", trees:(total))
exit()
```

Dentre estes comandos, o único que não foi utilizado anteriormente é o comando `set(root:"Taxon1")` – linha 2, que indica ao POY que todas as topologias deverão ser enraizadas pelo terminal `Taxon1`.

ii. Execute o *script* `meu_script1.poy` utilizandoo seguinte comando em seu terminal:

```
$ poy5.1.2a meu_script1.poy
```

Primeiro observe as últimas 31 linhas do registro de execução de POY. Você deve ter obtido:

... Status : Running Pipeline : 99 of 100 – Estimated finish in 0 s
Status : Wagner : 2 of 10 – Wagner tree with cost 30.
Status : Wagner : 3 of 10 – Wagner tree with cost 54.
Status : Wagner : 4 of 10 – Wagner tree with cost 66.
Status : Wagner : 5 of 10 – Wagner tree with cost 87.
Status : Wagner : 6 of 10 – Wagner tree with cost 113.
Status : Wagner : 7 of 10 – Wagner tree with cost 118.
Status : Wagner : 8 of 10 – Wagner tree with cost 139.
Status : Wagner : 9 of 10 – Wagner tree with cost 146.
Status : Wagner Finished
Status : Building Wagner Tree Finished
Status : Alternate : 0 Beginning search
Status : Alternate : 0 SPR search
Status : SPR : 158 158.
Status : SPR : 156 156.
Status : SPR Finished
Status : Alternate : 0 Performing TBR swapping
Status : Single TBR Finished
Status : Alternate Finished
Status : Running Pipeline : 100 of 100 – Estimated finish in 0 s
Status : Running Pipeline Finished
Status : Diagnosis : 0 Recalculating original tree

```
Status : Diagnosis Finished
Status : Diagnosis : 1 of 3 – Recalculating trees
Status : Diagnosis : 0 Recalculating original tree
Status : Diagnosis Finished
Status : Diagnosis : 2 of 3 – Recalculating trees
Status : Diagnosis : 0 Recalculating original tree
Status : Diagnosis Finished
Status : Diagnosis : 3 of 3 – Recalculating trees
Status : Diagnosis Finished
```

Quando POY executa um *script*, as instruções de execução funcionam de uma forma um pouco diferente em comparação com os comandos que você executou passo a passo anteriormente. No exemplo da seção 9.2.1, POY construiu todas as **RAS** para depois fazer o refinamento via SPR+TBR. Quando executado via *script*, POY faz o refinamento logo após a construção da árvore de Wagner e retém a(s) topologia(s) de menor custo. O resultado da análise pode ser verificado no arquivo `seqdata1_do.tre` que contém 3 topologias com o custo de 153:

```
(Taxon1,((Taxon4,(Taxon3,Taxon5)),((Taxon7,Taxon8),(Taxon9,(Taxon10,(Taxon2,Taxon6))))));
(Taxon1,((Taxon4,(Taxon3,Taxon5)),(Taxon10,((Taxon2,Taxon6),(Taxon9,(Taxon7,Taxon8))))));
(Taxon1,((Taxon4,(Taxon3,Taxon5)),((Taxon9,(Taxon2,Taxon6)),(Taxon10,(Taxon7,Taxon8)))));
```

Finalmente, vale ressaltar que a versão mais rescente de POY (atualmente 5.1.1) atribui custos idênticos para qualquer tipo de transformação (*i.e.*, substituições e INDELs) ao passo que nas versões anteriores a razão de custos entre INDELs:substituições é de 2:1.

iii. Os dados utilizados neste exercício são os mesmos que você utilizou no Tutorial 8 na seção 8.2. Compare os resultados de POY com aqueles obtidos por homologia estática registrados nas Tabelas 8.1 e 8.1 do Tutorial 8 e responda:

a. O custo obtido em POY é melhor ou pior do que os resultados que você havia obtido?

_____

_____

b. As topologias diferem?

_____

_____

9.2.3 Análises simultâneas em POY

Um dos componentes mais interessantes de POY é a possibilidade de analisar diferentes fontes de dados simultaneamente. Considere que você possui 3 partições com propriedades analíticas distintas. Suponha que você tenha uma partição que está sujeita a alinhamento, pois as sequências diferem de tamanho, uma outra partição cujos dados genotípicos provém de uma região codificadora que não contém INDELs, e ainda uma matriz de dados fenotípicos. Dentro do conceito de homologia estática, o procedimento analítico a ser adotado seria o alinhamento da primeira partição que posteriormente seria concatenada com as demais partições e este conjunto

de dados seria submetido à análise filogenética. No entanto, considere que o alinhamento da primeira partição depende de uma topologia para a qual as demais partições não possuem nenhuma influência. No exemplo que se segue iremos explorar como POY analisaria essas partições simultaneamente.

Considere os seguintes conjuntos de dados:

1. partition1.fas – dados não sujeitos a alinhamento
2. partition2.fas – dados sujeitos a alinhamento[3]
3. partition3.tnt – dados fenotípicos

Dentre as partições apresentadas acima, `partition1.fas` e `partition3.tnt` não devam ser submetidas á otimização direta ao passo que a `partition2.fas` deverá ser analisada por homologia dinâmica. A implementação de uma análise em POY que leve em consideração as propriedades destas partições requer a estrutura ilustrada no `script` abaixo (Script 9.2):

Listing 9.2: conteúdo do arquivo partitions_script.poy

```
read(prealigned:("partition1.fas"))
read("partition2.fas","partition3.tnt")
set(root:"Taxon1")
build(100)
swap()
select()
report(trees, treestats)
exit()
```

Este *script* está contido no arquivo `partitions_script.poy`. Na linha 1, POY lê os arquivos `partition1.fas` no qual o argumento `prealigned` do comando `read` informa ao POY que o arquivo `partition1.fas` contém sequências nucleotídicas pré-alinhadas. A linha 2, instrui o POY a ler os arquivos `partition2.fas` e `partition3.tnt` e considera que o primeiro será submetido à otimização direta ao passo que o segundo é uma matriz de dados de homologia estática – no caso, dados fenotípicos. Finalmente, o comando `report()` contém dois argumentos, `trees, treestats`, cujo primeiro retorna a(s) topologia(a) encontrada(s) e o segundo o custo da(s) mesma(s). Ao executá-lo você deverá obter:

```
...
Status : Wagner Finished
Status : Building Wagner Tree Finished
Status : Alternate : 0 Beginning search
Status : Alternate : 0 SPR search
Status : SPR : 376 376.
Status : SPR : 375 375.
```

[3] Estes dados encontram-se alinhados, porém POY irá desconsiderar os *gaps* a não ser que você especifique ao programa que esta partição deva ser considerada como pré-alinhada (veja abaixo)

```
Status : SPR Finished
Status : Alternate : 0 Performing TBR swapping
Status : Single TBR Finished
Status : Alternate Finished
Status : Running Pipeline : 100 of 100 – Estimated finish in 0 s
Status : Running Pipeline Finished
Information : Vectorized:8 of 8 characters.
Status : Diagnosis : 0 Recalculating original tree
Status : Diagnosis Finished
Status : Diagnosis : 1 of 1 – Recalculating trees
Status : Diagnosis Finished
(Taxon1,((Taxon5,(Taxon2,(Taxon4,Taxon9))),(Taxon6,(Taxon8,(Taxon3,(Taxon10,Taxon7))))))
;


Trees Found:


Tree length    Number of hits
370.           1
```

A análise destas partições resulta em uma única topologia com o custo de 370.

Exercicio 9.2

Neste exercício você deverá fazer uma análise de cada partição destes dados e comparar os resultados das análises individuais com a análise simultânea. Para cada análise, registre as topologias no espaço abaixo e responda:

Topologias encontradas nas análises filogenéticas das partições do Exercício 9.2:

Análise simultânea	Partição 1
Partição 2	Partição 3

i. As topologias são idênticas?

ii. Para cada clado presente na análise simultânea, identique se eles são corroborados pelas partições. Você considera que alguma partição tem maior influência na estrutura cladistica

(topologia) recuperada na análise simultânea?

_____

_____

iii. Considerando o número de caracteres de cada partição, existe alguma relação entre número de caracteres das partições e a influência na estutura da topologia da análise simultânea?

_____

_____

9.2.4 Implementação de matrizes de custo em POY

A implementação de parâmetros de alinhamento (= função de custo) em POY é feita pelo comando "`transform()`" e o argumento "`tcm:`". Este argumento define a matriz de custo de transformação para conjuntos de caracteres. O *default* de POY é `tcm:(1,1)`, no qual o custo para substituições e INDELs, respectivamente é 1. Em versões anteriores de POY, o custo de INDELs era 2, portanto, o argumento seria expresso como `tcm:(1,2)`. Para o exemplo anterior, há duas formas de implementar matrizes de custo de transformação. Na primeira delas, o Script 9.2 seria modificado da seguinte maneira:

Listing 9.3: Modificação do arquivo partitions_script.poy para implementar matriz de custo de transformação.

```
read(prealigned:("partition1.fas"))
read("partition2.fas","partition3.tnt")
transform(tcm:(1,2))
set(root:"Taxon1")
build(100)
swap()
select()
report(trees, treestats)
exit()
```

Neste *script* a linha 3 implementa os custos de transformação para todos os caracteres no qual o valor atribuído para substituições e INDELs é 1 e 2, respectivamente. No entanto, observe que somente uma das partições possuí INDELs e talvez seja desejável especificar para qual destas partições POY deverá considerar a matriz de custo em questão. Se modificarmos o Script 9.3 da seguinte maneira:

Listing 9.4: Modificação do arquivo partitions_script.poy para implementar matriz de custo de transformação para a partição `partition2.fas`.

```
read(prealigned:("partition1.fas"))
read("partition2.fas","partition3.tnt")
```

```
transform(names:("partition2.fas"),(tcm:(1,2)))
set(root:"Taxon1")
build(100)
swap()
select()
report(trees, treestats)
exit()
```

Neste caso, o argumento "`names:`" identifica a partição `partition2.fas` para qual a matriz de custo de transformação "`tcm:(1,2)`" será implementada.

Outra forma de implementar matrizes de custo em POY é utilizando arquivos que contém os custos de transformação. Estas matrizes são matrizes 5x5, no caso de sequências nucleotídicas, nas quais são especificadas os custos de transformação para cada estado (*i.e.*, `INDELs`, `A`, `C`, `G` e `T`). Por exemplo, verifique o conteúdo do arquivo 2111.tcm:

```
0 1 1 1 2
1 0 1 1 2
1 1 0 1 2
1 1 1 0 2
2 2 2 2 0
```

Esta matriz especifica as seguintes transformações:

```
    [A][C][G][T][-]
[A]  0   1   1   1   2
[C]  1   0   1   1   2
[G]  1   1   0   1   2
[T]  1   1   1   0   2
[-]  2   2   2   2   0
```

Essa é uma forma mais versátil de implementar matrizes de custo, pois possibilita que o usuário atribua custos diferenciais para determinados tipos de transformação (*e.g.*, transições e transversões). A linha de comando que implementa matrizes de custo expressas em arquivos com esse conteúdo é ilustrado na modificação do Script 9.5 feita abaixo:

Listing 9.5: Modificação do arquivo partitions_script.poy para implementar matriz de custo de transformação para a partição `partition2.fas`.

```
read(prealigned:("partition1.fas"))
read("partition2.fas","partition3.tnt")
transform(names:("partition2.fas"),(tcm:("211.tcm")))
set(root:"Taxon1")
```

```
build(100)
swap()
select()
report(trees, treestats)
exit()
```

Exercicio 9.3

Neste exercício você deverá executar 2 análises em POY para os dados contidos nos arquivos `partition1.fas`, `partition2.fas` e `partition3.tnt` em que o custo para INDELs seja 2 e 4. As topologias resultantes destas análises deverão ilustradas no espaço abaixo e você deverá comentar se os resultados diferem da análise anterios na qual os custos para INDELs e substituições eram idênticos.

`tcm:(1,2)`	`tcm:(1,4)`

9.3 Referências

1. Wheeler, W. C. 2012. Systematics: a course of lectures. Malaysia: Wiley-Backwell, 2012. 426.
2. Ogden, T. H. & Rosemberg, M. S. 2007. Alignment and topological accuracy of the direct optimization approach via POY and traditional phylogenetics via Clustalw + PAUP*. *Systematic Biology* **56**: 182–193.
3. Wheeler, W. C. & Giribet, G. em *Sequence Alignment: Methods, models, concepts and strategies* ed. Rosemberg, M. S., 337. Berkeley, CA: University of California Press, 2009.

4. Sankoff, D. 1975. Minimal mutation trees of sequence. *SIAM Journal on Applied Mathematics* **28**: 35–42.

5. Wheeler, W. C. 2003. Implied alignment: a synapomorphy-based multiple-sequence alignment method and its use in cladogram search. *Cladistics* **19**: 261–268.

6. Wheeler, W. C. 2001. Homology and the optimization os DNA sequence data. *Cladistics* **17**: S3–S11.

7. Wheeler, W. C. 1996. Optimization alignment: the end of multiple sequence alignment in phylogenetics? *Cladistics* **12**: 1–9.

8. Varón, A.; Vinh, L. S. & Wheeler, W. C. 2010. POY version 4: phylogenetic analysis using dynamic homologies. *Cladistics* **26**: 72–85.

9. Varon, A.; Lucaroni, N.; Hong, L. & Wheeler, W. C. 2011–2014. POY version 4: phylogenetic analysis using dynamic homologies, version 5.0. New York, NY: American Museum of Natural History, 2011–2014.

10. Needleman, S. B. & Wunsch, C. D. 1970. A general method applicable to the search of similatiries in the amino acid sequence of two proteins. *Journal of Molecular Biology* **48**: 443–153.